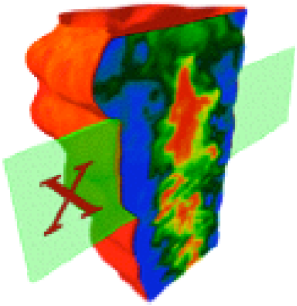


OMA
Software Developers Kit

Documentation

1 Introduction

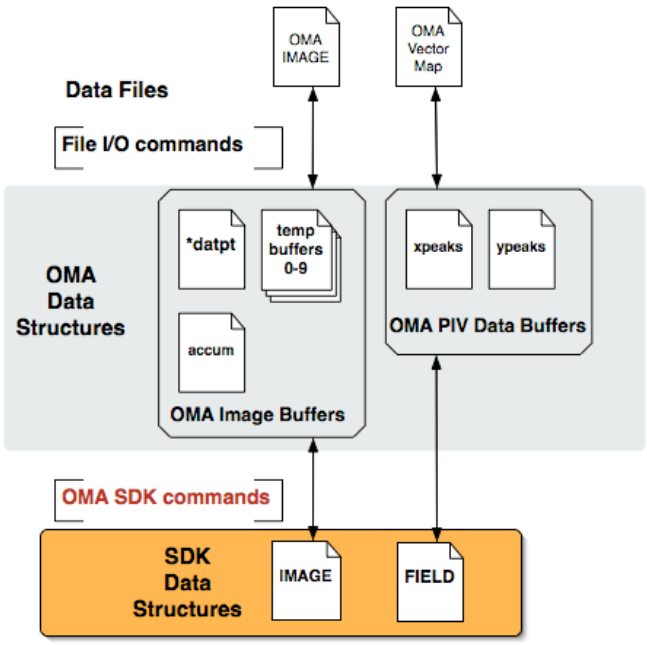
The OMA Software Developers Kit (SDK) has been written as an aide to those users that need to expand the existing OMA functionality. The source code for OMA is available under the GNU Public License, and the Developer's bundle for OS X contains the OMA Project file for Apple's XCode 2.1 compiler. This compiler and Integrated Developers Environment (IDE) is available for download from the Apple Developers Connections (ADC) website.

The SDK is also released under the GNU Public License, in the interests of serving the OMA 'family', by its author Peter Kalt.

Peter Kalt is a Senior Research Fellow at the University of Adelaide, and a long-time OMA contributor and user. He uses OMA for the processing of photometric and velocity data collected in laser imaging experiments in turbulent non-reacting flows and flames.

2 Using the SDK

The SDK provides a framework of functions, written in C, to serve as a convenient interface between the programmer and the data structures underlying the OMA functionality. Specifically, the SDK frees the programmer from many tedious tasks in managing data structures and accessing stored information, manipulating headers and footers, etc.



When a developer needs to write some specific image processing functionality not already implemented in OMA, there is the option to write a new OMA command. The following steps outline how this is done.

1) Open CUSTOM.C

It is not recommended to add functions to the implementation files within OMA. These should be left untouched. Instead, new functions should be added to the file CUSTOM.c. This file is the repository for all custom commands in OMA, and already has the suitable headers needed to access the SDK.

2) Add Functional PROTOTYPE

Add the function prototype at the beginning of the file, amongst the other similar declarations. It is suggested that alphabetical order of the function names are preserved. If the function we are creating were to be called `New_OMA_Function`, then we would add the following:

```
int New_OMA_Function();
```

[NB: The custom function should return an `int` not `void`. It is expected that all OMA Command line functions return an integer value, which should correspond to a defined return state, indicating that the function executed successfully (or not). These return values are defined in the header (custom.h) and are described in Section 4 below.]

3) Add COMMAND HANDLER

First step in adding a new OMA command is to think of a 6-character command name that can be parsed from the command line. The command name should be descriptive, unique and should not contain spaces or special characters.

You can test whether or not your new command name is suitable, by typing it in the command window of OMA whilst it is running. If the command is unique and suitable, and not yet used, you will get the following error:

```
OMA>NEWOMA
No such command: NEWOMA
OMA>
```

Add the command to the command handler at the beginning of the custom.c file. Following our example, the function `New_OMA_Function()`, will be called using the command name `NEWOMA`. Add the command name and its function to the command list in alphabetical order.

```
{{"NEWOMA"}, New_OMA_Function},
```

[NB: It is IMPORTANT that the command name is capitalised in this list. Internally, all input to the command window is capitalised before use, even if it is entered in lower case.]

[NB: It is *possible* to have a 5-character OMA command name, but the trailing space must be included in the string within the quotes.]

```
{{"5CHAR_"}, Another_OMA_Function},
```

4) Add the new function at the end of CUSTOM.c

If the function requires command line parameters:

```
int New_OMA_Function(int index, int n)
{return OMA_OK;}
```

Otherwise,

```
int New_OMA_Function()
{return OMA_OK;}
```

is sufficient.

5) Use the SDK to get access to the OMA data in the image buffer or temporary buffers.

• **Allocate Image data structures in the function.** The Image and Field data structures are used to store a 2-D array of pixels images and 3-component vectors, respectively. Each instance of an Image or Field needs to be allocated within the function.

```
Image myImage;
Field myField;
```

• **Call the SDK functions by reference**.

```
SDK_Image_Copy(&myImage);
```

• **Manipulate the IMAGE data structures.** you have defined using the SDK, rather than the OMA data structures.

```
for (I=0;I<width;I++){
    for (J=0; J<height; J++){
      SDK_Image_SetPixel(&myImage, I, J, 3.1417);
    }
}
```

• Once you have finished your processing, **transfer your data structures back to OMA** using the SDK.

```
SDK_Image_2Buffer(&myImage);
```

• Finally, **refresh the OMA data structures and clean up the memory** that has been used.

```
SDK_Image_Free(&myImage);
SDK_Image_Refresh();
```

3 SDK Functions

SDK_Image_Copy

```
int SDK_Image_Copy (Image *imptr)
```

Input:	`*imptr`	A pointer to an Image structure
Returns:	`int`	OMA Error Code

Copies the image data currently in the OMA image buffer to an `Image` structure.

SDK_Image_CopyTemp

```
int SDK_Image_CopyTemp (Image *imptr, int n)
```

Input:	`*imptr`	A pointer to an Image structure
	`n`	The temporary image buffer number (0-9)
Returns:	`int`	OMA Error Code

Copies the image data currently in the OMA temporary buffer #n to an `Image` structure.

SDK_Image_Create

```
int SDK_Image_Create (Image *imptr, short width, short height)
```

Input:	`*imptr`	A pointer to an Image structure
	`width`	Width of the new image
	`height`	Height of the new image
Returns:	`int`	OMA Error Code

Creates a new `Image` structure with the specified `width` and `height`. The new Image structure has pixel values initialized to zero.

SDK_Image_Free

```
int SDK_Image_Free (Image *imptr)
```

Input:	`*imptr`	A pointer to an Image structure
Output:	`int`	OMA Error Code

Release the memory used by an `Image` structure.

SDK_Image_Refresh

```
int SDK_Image_Copy (Image *imptr)
```

Input:	NONE	
Output:	int	OMA Error Code

Refresh the properties of the OMA image buffers after we have been touching them.

SDK_Image_2Buffer

```
int SDK_Image_2Buffer (Image *imptr)
```

Input:	*imptr	A pointer to an Image structure
Output:	int	OMA Error Code

Moves the data in a SDK `Image` structure to the OMA Image buffer.

SDK_Image_2Temp

```
int SDK_Image_2Temp (Image *imptr, int n)
```

Input:	*imptr	A pointer to an Image structure
	n	The temporary image buffer number (0-9)
Output:	int	OMA Error Code

Moves the data in a SDK `Image` structure to the OMA temporary image buffer #n.

SDK_Image_SetPixel

```
int SDK_Image_SetPixel (Image *imptr, short xpos,
                        short  ypos, DATAWORD value)
```

Input:	*imptr	A pointer to an Image structure
	xpos	X position of the pixel to be changed
	ypos	y position of the pixel to be changed
	value	new value of the pixel (xpos, ypos)
Output:	int	OMA Error Code

Sets pixel (`xpos`, `ypos`) in the SDK `Image` structure to `value`.

SDK_Image_GetPixel

```
DATAWORD SDK_Image_GetPixel (Image *imptr, short xpos,
                             short  ypos)
```

Input:	*imptr	A pointer to an Image structure
	xpos	X position of the pixel to be changed
	ypos	y position of the pixel to be changed
Output:	DATAWORD	Function returns the value of the pixel

Gets the value of pixel (xpos, ypos) in the SDK Image.

SDK_Image_InterpPixel

```
DATAWORD SDK_Image_GetPixel (Image *imptr, float x,
                             float  y)
```

Input:	*imptr	A pointer to an Image structure
	x	X position of the pixel to be changed
	y	y position of the pixel to be changed
Output:	DATAWORD	Function returns the value of the interpolated pixel value

Returns the bilinear interpolated value of the pixel around (x, y) in the SDK Image structure. The position is given by floats (x, y) which can represent fractional values. 8-parameter bilinear interpolation is used.

SDK_Image_GetMaxPixel

SDK_Image_GetMinPixel

```
DATAWORD SDK_Image_GetMaxPixel (Image *imptr)
DATAWORD SDK_Image_GetMinPixel (Image *imptr)
```

Input:	*imptr	A pointer to an Image structure
Output:	DATAWORD	Function returns a pixel value.

These functions returns the value of the maximum/minimum pixel in the SDK Image.

SDK_Image_GetMaxPixelX

SDK_Image_GetMinPixelX

```
int SDK_Image_GetMaxPixelX (Image *imptr)
int SDK_Image_GetMinPixelX (Image *imptr)
```

Input:	*imptr	A pointer to an Image structure
Output:	int	Function returns position of pixel in row.

These functions returns the x-position of the maximum/minimum pixel in the SDK Image.

SDK_Image_GetMaxPixelY

SDK_Image_GetMinPixelY

```
int SDK_Image_GetMaxPixelY (Image *imptr)
int SDK_Image_GetMinPixelY (Image *imptr)
```

Input:	*imptr	A pointer to an Image structure
Output:	int	Function returns position of pixel in column.

These functions returns the y-position of the maximum/minimum pixel in the SDK Image.

4 OMA Return values

The following values are defined in custom.c and should be used to indicate a successful return from the custom command, or to assist in determining which error caused failure. Error handling is the responsibility of the programmer, however.

```
#define OMA_OK          0
```
Indicates that the command completed successfully.

```
#define OMA_MEMORY     -1
```
Indicates that the command failed due to running out of memory, or failing to allocate new memory.

```
#define OMA_FILE       -2
```
Indicates that a OMA was unable to open a file it needed. This could be because the path set in the control panel is not set correctly.

```
#define OMA_RANGE      -4
```
This error indicates that OMA was passed a value out of range. Normally, OMA should try to compensate and correct for range issues, such as asking for the value of a pixel outside the bounds of the image. Sometimes, it is better to simply end the function and leave with this error, such as trying to access a TEMPORARY image buffer # greater than those allowed by OMA.

```
#define OMA_NOEXIST    -5
```
This error is used to indicate that OMA was unable to find something it had assumed was there.

```
#define OMA_MISC       -6
```
This OMA error code is used for general errors that are not more suitably described by any other code.

```
#define OMA_ARGS       -7
```
This error is used when OMA a command has been passed the wrong number of command line arguments and is unable to substitute defaults.

5 Examples

```
//*******************************************************//
//**    FLIPH - Flip the image horizontally              *//
//*******************************************************//
int
fliph ()
{
        int nc,nt;
        Image Im_p;

        if(SDK_Image_Create(&Im_p,header[NCHAN],header[NTRAK])!=OMA_OK){
                beep();
                printf("SDK_Image_Create returned an error\n");
                return OMA_MEMORY;
        }

        for(nt=0; nt<header[NTRAK]; nt++) {
                for (nc =0; nc<header[NCHAN]; nc++){
                        SDK_Image_SetPixel(&Im_p,header[NCHAN]-1-nc, nt,
                                                PKGetPixel(nc,nt));
                }
        }
        SDK_Image_2Buffer(&Im_p);
        SDK_Image_Free(&Im_p);
        SDK_Refresh();
        return OMA_OK;
}

//*************************************************************//
//**    FLIPV - Flip the image horizontally                  *//
//*************************************************************//
int
flipv ()
{
        int nc,nt;
        Image myimage;

        if(SDK_Image_Create (&myimage, header[NCHAN],header[NTRAK])!=OMA_OK){
                beep();
                nomemory();
                return OMA_MEMORY;
        }

        for(nt=0; nt<header[NTRAK]; nt++) {
                for (nc =0; nc<header[NCHAN]; nc++){
                        SDK_Image_SetPixel(&myimage, nc,header[NTRAK]-1- nt,
PKGetPixel(nc,nt));
                }
        }
        SDK_Image_2Buffer(&myimage);
        SDK_Image_Free(&myimage);
        SDK_Refresh();
        return OMA_OK;
}
```

```
//*************************************************************************//
//**   POLARN - Custom function by pkalt used for polaristion processing  *//
//*************************************************************************//
int
polarn (int n, int index)
{
        int nc,nt;
        Image NumSamp;
        Image Result;
        Image Im_Buff;
        Image T0_Buff;
        int width, height;
        DATAWORD numerator, denominator;
        DATAWORD p_ratio, tempval;

        // Copy the denominator  from the file in Temp_Image = 0
        if(SDK_Image_CopyTemp(&T0_Buff, 0)!=OMA_OK){
                beep();
                printf("Need to have the denominator in temp buffer 0\n");
                return OMA_MEMORY;
        }

        // Create an image with from the file in Buffer
        if(SDK_Image_Copy(&Im_Buff)!=OMA_OK){
                beep();
                printf("SDK_Image_Copy() returned an error\n");
                return OMA_MEMORY;
        }
        // Get the dimensions of this image for the other new ones
        width = header[NCHAN];
        height = header[NTRAK];

        // Create new image structures for Result
        if(SDK_Image_Create(&Result, width, height)!=OMA_OK){
                beep();
                printf("SDK_Image_Create() returned an error\n");
                return OMA_MEMORY;
        }

        // If there is no existing NumSamp in temp buffer 2 create,
        // otherwise use it
        if(SDK_Image_CopyTemp(&NumSamp, 2)!=OMA_OK){
                if (SDK_Image_Create(&NumSamp, width, height)!=OMA_OK){
                        beep();
                        printf("SDK_Image_Create() returned an error\n");
                        return OMA_MEMORY;
                }
        }

        // For each row and column process each pixel in turn...
        for(nt=0; nt<height; nt++) {
                for (nc =0; nc<width; nc++){
                        numerator = SDK_Image_GetPixel(&Im_Buff, nc, nt);
                        denominator = SDK_Image_GetPixel(&T0_Buff, nc, nt);
                        if (denominator!=0){
                                p_ratio = numerator/denominator;
                                SDK_Image_SetPixel(&Result, nc, nt, p_ratio);
                                tempval = SDK_Image_GetPixel(&NumSamp, nc, nt);
                                SDK_Image_SetPixel(&NumSamp, nc, nt, tempval+1);
                        } else {
                                SDK_Image_SetPixel(&Result, nc, nt,
```

```
                        (DATAWORD)1.000000);
                }
            }
        }

        // Once done, get ready to leave
        SDK_Image_2Buffer(&Result);
        //SDK_Image_2Temp(&Result, 1);
        SDK_Image_2Temp(&NumSamp, 2);

        // Free the old Image data structures
        SDK_Image_Free(&Im_Buff);
        SDK_Image_Free(&Result);
        SDK_Image_Free(&TO_Buff);
        SDK_Image_Free(&NumSamp);

        SDK_Refresh();
        return OMA_OK;
}
```

6 License